software vendors. The network services API 240 makes the hardware-specific functionality of the data forwarding elements 210 (such as switches, routers, and network interface cards (NICs)) available to the application programmers in a uniform, hardware-independent manner. Third party independent software vendors that write network-aware applications, such as VoIP (voice-over internet protocol) gateways, intrusion detection, application-specific proxies, and VPN (virtual private network) servers, use the network service API 240 to control/modify the behavior of the data forwarding path in both the data and control planes. For example, in the case of a L3/L4 switch with the ability to filter packets based on pre-specified packet filters, an H.323 (H.323, Packet-Based Multimedia Communication Systems, International Telecommunication Union, February 1998) proxy uses the network services API 240 to direct the forwarding elements 210 to intercept and forward H.323-related packets to itself for further processing. However, in the case of a L4/L7 switch with the capability of stateful inspection of data packets at wire-speed, an intrusion detection application could install a policy rule in the forwarding element 210 that specifies the classification filter and the associated actions for examining session state and identifying an "intrusion signature". In the horizontal open networking architecture 200 according to an embodiment of the present invention, the control element 230 hosts the network services API 240 implementation and hides the details of translating the network services API 240 calls to appropriate message passing and invocations of hardware-specific calls at the forwarding element 210.

The paragraph starting at page 14, line 7 has been amended as follows:

The connect API 220 itself includes the basic communication primitives along with methods for examining and manipulating the status of the underlying transport in a transport independent manner. The connect API 220 may be implemented over a variety of transports, such as PCI (peripheral component interconnect), input/output backplane, Ethernet, or ATM (asynchronous transfer mode). Depending on the transport, a transport-specific module specifies exactly how the connect API UDPs (user datagram protocols) are encapsulated on specific interconnect technologies, as well as how to deal with normal and exception conditions in the operation of the interconnect technology. Two examples of transports include PCI and IP. The IP transport is preferably used over the Ethernet or ATM, and can be implemented using either UDP (user datagram protocol) or TCP (transmission control protocol) as transport protocols. The use of these protocols, however, does not preclude the possibility of using native ATM or Ethernet transport for implementation of the connect API 220.

The paragraph starting at page 16, line 13 has been amended as follows:

The forwarding element-specific configuration BLOB (specialized data set) 495, contains the forwarding element-specific invocations of functionality that is specific to the proprietary forwarding element 210. By utilizing the forwarding element-specific plugin 490 to translate a standardized data set into a specialized data set, the proprietary forwarding element 210 implementations need not be exposed. Therefore, the proprietary design and architecture information are kept confidential. Because the forwarding element-specific plugin 490 is in binary form, the transformation process from the standardized data set to the specialized data set is essentially hidden from

other components, thus protecting the proprietary forwarding element 210 implementation. A tremendous amount of complex reverse engineering would be required in order to determine the translation process from the forwarding element specific plugin DLL file.

The paragraph starting at page 17, line 1 has been amended as follows:

Once the forwarding element-specific configuration BLOB 495 is generated, it is preferably passed back to the opaque forwarding element plugin API 480, which then transmits the BLOB 495 to the proprietary forwarding element 210. The BLOB 495 is preferably passed through the open forwarding element/control element interconnect 220, then to the abstract forwarding element API 440, and then finally to the BLOB decapsulator 430. The BLOB decapsulator 430 takes the BLOB 495 and "decapsulates" it — which is a very simple computational operation — and passes the decapsulated BLOB data directly to the device-specific forwarding element interface 420. The device-specific forwarding element interface 420 takes the information from the decapsulated BLOB data to configure the forwarding element software and hardware 410 to properly operate the proprietary forwarding element 210. In this manner, confidential information about the proprietary forwarding element's architecture that is present in the device-specific forwarding element interface 420 is never exposed to the standard control element 230, allowing independent hardware vendors of the forwarding elements to protect their intellectual property.